Python for Data Science

General Assembly, NY

Something to start with

With your best knowledge, write a python script that:

1. loads a file from a website (`http://stat.columbia.edu/~rachel/datasets/nyt1.csv`)
2. from that file, counts the number of 1s and 0s under the 'Gender' column, and the number of 1s and 0s under the 'Signed_In' column

A Simple Solution

Read the script below.

```
## Download the file into ~/Downloads
## run: python nytimes_counter.py < ~/Downloads/nyt1.csv
# Import required libraries
import sys

# Start a counter and store the textfile in memory
gender = 0
signins = 0
lines = sys.stdin.readlines()
lines.pop(0)

# For each line, find the sum of index 2 in the list.
for line in lines:

  gender = gender + int(line.strip().split(',')[1])

for line in lines:
  signins = signins + int(line.strip().split(',')[4])

gender_0 = len(lines) - gender
signins_0 = len(lines) - signins

print "Gender 0: ", gender_0
print "Gender 1: ", gender_1
print "Signin 0: ", signins_0
print "Signin 1: ", signins_1
```

Optimizations

Download in terminal, or python

On unix machines, we can use either curl or wget to download the file:

```
curl http://stat.columbia.edu/~rachel/datasets/nyt1.csv > nyt1.csv
wget http://stat.columbia.edu/~rachel/datasets/nyt1.csv > nyt1.csv
```

on PC, you can use powershell:

```
Invoke-WebRequest http://stat.columbia.edu/~rachel/datasets/nyt1.csv -OutFile
 nyt1.csv
```

since we're going all the data manipulation in python, we could also use a library to store the data in memory. Below are two common approaches to this.

```
import csv
import requests
import urllib2
import StringIO

url = 'http://stat.columbia.edu/~rachel/datasets/nyt1.csv'

## urllib2 version
response = urllib2.urlopen(url)
nyt = csv.reader(response)



## requests version
r = requests.get(url)
data = r.text
nyt = csv.reader(data.splitlines(), delimiter='\t')
```

Loop once if you only have to loop once

One way to improve the script above is to loop through the iterator `nyt` only once:

```
for line in nyt:
    gender = gender + line[1]
    signins = signins + line[4]
```

New Script

In [2]:
```
import csv
import requests
import urllib2
import StringIO

url = 'http://stat.columbia.edu/~rachel/datasets/nyt1.cs
v'

response = urllib2.urlopen(url)
nyt = csv.reader(response)

counts, gender, signins = 0, 0, 0

# ignores the header row
next(nyt, None)
for line in nyt:
    counts += 1
    gender += int(line[1])
    signins += int(line[4])

gender_0 = counts - gender
signins_0 = counts - signins

print "Gender 0:", gender_0
print "Gender 1:", gender
print "Signin 0:", signins_0
print "Signin 1:", signins
```

```
Gender 0: 290176
Gender 1: 168265
Signin 0: 137106
Signin 1: 321335
```

Learning Python in an Hour

adapted from Alysaa Frazee, introducing R to a non-programmer in one hour (http://alyssafrazee.com/introducing-R.html)

In [19]:
```python
x = 7
print x + 5
# These are comments! And they are super helpful!
# use help() to get help about what something is doing
help(x)
```

```
12
Help on int object:

class int(object)
 |  int(x=0) -> int or long
 |  int(x, base=10) -> int or long
 |
 |  Convert a number or string to an integer, or return 0
 if no arguments
 |  are given.  If x is floating point, the conversion tr
uncates towards zero.
 |  If x is outside the integer range, the function retur
ns a long instead.
 |
 |  If x is not a number or if base is given, then x must
 be a string or
 |  Unicode object representing an integer literal in the
 given base.  The
 |  literal can be preceded by '+' or '-' and be surround
ed by whitespace.
 |  The base defaults to 10.  Valid bases are 0 and 2-36.
  Base 0 means to
 |  interpret the base from the string as an integer lite
ral.
 |  >>> int('0b100', base=0)
 |  4
 |
 |  Methods defined here:
 |
 |  __abs__(...)
 |      x.__abs__() <==> abs(x)
 |
 |  __add__(...)
 |      x.__add__(y) <==> x+y
 |
 |  __and__(...)
 |      x.__and__(y) <==> x&y
 |
 |  __cmp__(...)
```

```
|      x.__cmp__(y) <==> cmp(x,y)
|
|  __coerce__(...)

|      x.__coerce__(y) <==> coerce(x, y)
|
|  __div__(...)
|      x.__div__(y) <==> x/y
|
|  __divmod__(...)
|      x.__divmod__(y) <==> divmod(x, y)
|
|  __float__(...)
|      x.__float__() <==> float(x)
|
|  __floordiv__(...)
|      x.__floordiv__(y) <==> x//y
|
|  __format__(...)
|
|  __getattribute__(...)
|      x.__getattribute__('name') <==> x.name
|
|  __getnewargs__(...)
|
|  __hash__(...)
|      x.__hash__() <==> hash(x)
|
|  __hex__(...)
|      x.__hex__() <==> hex(x)
|
|  __index__(...)
|      x[y:z] <==> x[y.__index__():z.__index__()]
|
|  __int__(...)
|      x.__int__() <==> int(x)
|
|  __invert__(...)
|      x.__invert__() <==> ~x
|
|  __long__(...)
|      x.__long__() <==> long(x)
|
|  __lshift__(...)
|      x.__lshift__(y) <==> x<<y
|
```

```
 |
 |  __mod__(...)
 |      x.__mod__(y) <==> x%y
 |
 |  __mul__(...)
 |      x.__mul__(y) <==> x*y
 |
 |  __neg__(...)
 |      x.__neg__() <==> -x
 |
 |  __nonzero__(...)
 |      x.__nonzero__() <==> x != 0
 |
 |  __oct__(...)
 |      x.__oct__() <==> oct(x)
 |
 |  __or__(...)
 |      x.__or__(y) <==> x|y
 |
 |  __pos__(...)
 |      x.__pos__() <==> +x
 |
 |  __pow__(...)
 |      x.__pow__(y[, z]) <==> pow(x, y[, z])
 |
 |  __radd__(...)
 |      x.__radd__(y) <==> y+x
 |
 |  __rand__(...)
 |      x.__rand__(y) <==> y&x
 |
 |  __rdiv__(...)
 |      x.__rdiv__(y) <==> y/x
 |
 |  __rdivmod__(...)
 |      x.__rdivmod__(y) <==> divmod(y, x)
 |
 |  __repr__(...)
 |      x.__repr__() <==> repr(x)
 |
 |  __rfloordiv__(...)
 |      x.__rfloordiv__(y) <==> y//x
 |
 |  __rlshift__(...)
 |      x.__rlshift__(y) <==> y<<x
 |
```

```
 |
 |  __rmod__(...)
 |      x.__rmod__(y) <==> y%x
 |
 |  __rmul__(...)
 |      x.__rmul__(y) <==> y*x
 |
 |  __ror__(...)
 |      x.__ror__(y) <==> y|x
 |
 |  __rpow__(...)
 |      y.__rpow__(x[, z]) <==> pow(x, y[, z])
 |
 |  __rrshift__(...)
 |      x.__rrshift__(y) <==> y>>x
 |
 |  __rshift__(...)
 |      x.__rshift__(y) <==> x>>y
 |
 |  __rsub__(...)
 |      x.__rsub__(y) <==> y-x
 |
 |  __rtruediv__(...)
 |      x.__rtruediv__(y) <==> y/x
 |
 |  __rxor__(...)
 |      x.__rxor__(y) <==> y^x
 |
 |  __str__(...)
 |      x.__str__() <==> str(x)
 |
 |  __sub__(...)
 |      x.__sub__(y) <==> x-y
 |
 |  __truediv__(...)
 |      x.__truediv__(y) <==> x/y
 |
 |  __trunc__(...)
 |      Truncating an Integral returns itself.
 |
 |  __xor__(...)
 |      x.__xor__(y) <==> x^y
 |
 |  bit_length(...)
 |      int.bit_length() -> int
 |
```

```
 |
 |      Number of bits necessary to represent self in bin
ary.
 |      >>> bin(37)
 |      '0b100101'
 |      >>> (37).bit_length()
 |      6
 |
 |  conjugate(...)
 |      Returns self, the complex conjugate of any int.
 |
 |  ----------------------------------------------------
-----------------
 |  Data descriptors defined here:
 |
 |  denominator
 |      the denominator of a rational number in lowest te
rms
 |
 |  imag
 |      the imaginary part of a complex number
 |
 |  numerator
 |      the numerator of a rational number in lowest term
s
 |
 |  real
 |      the real part of a complex number
 |
 |  ----------------------------------------------------
-----------------
 |  Data and other attributes defined here:
 |
 |  __new__ = <built-in method __new__ of type object>
 |      T.__new__(S, ...) -> a new object with type S, a
subtype of T
```

Basic Data Types

In [24]:
```python
# Strings, Lists, Tuples

some_string1 = 'apples'
some_string2 = 'and'
some_string3 = 'bananas'
print some_string1, some_string2, some_string3

mutable_list = ["apple", "apple", "banana", "kiwi", "be
ar", "strawberry", "strawberry"]
immutable_tuple = ("apple", "apple", "banana", "kiwi",
"bear", "strawberry", "strawberry")

print len(some_string1)
print len(mutable_list)
print len(immutable_tuple)

print some_string1[0:5]
print mutable_list[0:4]
print immutable_tuple[5:6]

#some_string1[5] = 'd'
mutable_list[5] = 'mango'
#immutable_tuple[5] = 'not going to work'

a = [3 for i in range(100)]
print a
```

```
apples and bananas
6
7
7
apple
['apple', 'apple', 'banana', 'kiwi']
('strawberry',)
[3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
 3, 3, 3, 3, 3]
```

Dictionaries, functions

In [45]:
```
some_dictionary = dict()
some_dictionary['fruits'] = ['apples', 'oranges', 'bana
nas']
some_dictionary['veggies'] = ['beans', 'carrots', 'kale
', 'beats']
print some_dictionary['fruits']
print some_dictionary.keys()
print some_dictionary.values()
print some_dictionary.items()
print

def print_items(dictionary):
    for k,v in dictionary.iteritems():
        print '%s: %s' % (k, v,)

def longest_list(dictionary):
    return max(enumerate(dictionary.values()), key = la
mbda tup: len(tup[1]))

print_items(some_dictionary)

print longest_list(some_dictionary)
```

```
['apples', 'oranges', 'bananas']
['veggies', 'fruits']
[['beans', 'carrots', 'kale', 'beats'], ['apples', 'orang
es', 'bananas']]
[('veggies', ['beans', 'carrots', 'kale', 'beats']), ('fr
uits', ['apples', 'oranges', 'bananas'])]

veggies: ['beans', 'carrots', 'kale', 'beats']
fruits: ['apples', 'oranges', 'bananas']
(0, ['beans', 'carrots', 'kale', 'beats'])
```

Classes

Python is considered Object Oriented Programming (OOP), and because of that, much of the functionality with libraries used in Python and in the Data Science course will be Classes. Consider the following changes to the script above:

```
In [185]: import csv
          import requests
          import urllib2
          import StringIO

          class NYTimesCounter():

              def __init__(self):
                  # initializes the object
                  self.counts = 0
                  self.gender = 0
                  self.signin = 0

              def reset(self):
                  # Resets the counters
                  self.counts = 0
                  self.gender = 0
                  self.signin = 0

              def _get_url(self, url):
                  # private function that retrieves the file,
                  # and if the url is local, retrieves locally i
          nstead.
                  if url[0:4] == 'http':
                      self.response = urllib2.urlopen(url)
                  else:
                      self.response = open(url)

              def run_file(self, url):
                  # calls get_url() and counts the data
                  self.reset()
                  self._get_url(url)
                  nyt = csv.reader(self.response)
                  next(nyt, None)
                  for line in nyt:
                      self.counts += 1
                      self.gender += int(line[1])
                      self.signin += int(line[4])
```

```
        def print_params(self):
            print "Gender 0:", self.gender
            print "Gender 1:", self.counts - self.gender
            print "Signin 0:", self.signin
            print "Signin 1:", self.counts - self.gender

    nyt = NYTimesCounter()
    print type(nyt)
    #nyt.run_file('http://stat.columbia.edu/~rachel/datase
    ts/nyt1.csv')
    nyt.run_file('/Users/edjoy/Downloads/nyt1.csv')
    nyt.print_params()
```

```
<type 'instance'>
Gender 0: 168265
Gender 1: 290176
Signin 0: 321335
Signin 1: 290176
```

Takeaways

1. How is the above script different from what we ran previously?
2. What advantages and disadvantages do you see in OOP vs functional programming (which is similar to the earlier function)

Libraries for Data Science

Data Scientists use a wide variety of libraries in Python that make working with data significantly easier. Those libraries primarily consist of:

1. `numpy`
2. `scipy`
3. `pandas`
4. `matplotlib`
5. `statsmodels`
6. `scikit-learn`
7. `nltk`

Though there are countless others available.

For today, we'll primarily focus ourselves around `pandas`, `matplotlib`, and `sklearn`.

`pandas`

`pandas` is a library built on top of `numpy`, which allows us to use excel-like matrices in the python programming space. These special matrices are called DataFrames.

Load up the `nyt1.csv` (even from the website directly) as a pandas DataFrame like so:

In [186]:
```
import pandas as pd

#nyt = pd.read_csv('http://stat.columbia.edu/~rachel/datasets/nyt1.csv')
nyt = pd.read_csv('/Users/edjoy/Downloads/nyt1.csv')
```

In [187]:
```
print type(nyt)
print nyt.dtypes

print nyt.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
Age              int64
Gender           int64
Impressions      int64
Clicks           int64
Signed_In        int64
dtype: object
                 Age         Gender    Impressions
  Clicks  \
count  458441.000000  458441.000000  458441.000000  45844
1.000000
mean       29.482551       0.367037       5.007316
0.092594
std        23.607034       0.481997       2.239349
0.309973
min         0.000000       0.000000       0.000000
0.000000
25%         0.000000       0.000000       3.000000
0.000000
50%        31.000000       0.000000       5.000000
0.000000
75%        48.000000       1.000000       6.000000
0.000000
max       108.000000       1.000000      20.000000
4.000000

           Signed_In
count  458441.000000
mean        0.700930
std         0.457851
min         0.000000
25%         0.000000
50%         1.000000
75%         1.000000
max         1.000000
```

The describe function for a data frame creates a high level view of what your data looks like. With quantile distributions plus a mean (average), We can measure the same information we had before (.701 * 458441 = ~321360 Gender 0), though we can use other built in functions for this as well.

Like dictionaries, you call on columns using their keys, and like lists, you can subset on indices.

In [3]:
```
print nyt['Gender'].sum()
print nyt['Signed_In'].sum()

print nyt[1:3]
print nyt.head()
print nyt.head(10)

print nyt.tail()
```

```
168265
321335
   Age  Gender  Impressions  Clicks  Signed_In
1   73       1            3       0          1
2   30       0            3       0          1
   Age  Gender  Impressions  Clicks  Signed_In
0   36       0            3       0          1
1   73       1            3       0          1
2   30       0            3       0          1
3   49       1            3       0          1
4   47       1           11       0          1
   Age  Gender  Impressions  Clicks  Signed_In
0   36       0            3       0          1
1   73       1            3       0          1
2   30       0            3       0          1
3   49       1            3       0          1
4   47       1           11       0          1
5   47       0           11       1          1
6    0       0            7       1          0
7   46       0            5       0          1
8   16       0            3       0          1
9   52       0            4       0          1
        Age  Gender  Impressions  Clicks  Signed_In
458436    0       0            2       0          0
458437    0       0            4       0          0
458438   72       1            5       0          1
458439    0       0            5       0          0
458440    0       0            3       0          0
```

DataFrames allow you to subset as well. What does the data look like if you subset this click and impression data by Signed_In?

In [4]:
```
print nyt[nyt['Signed_In'] == 0].describe()

# groupby is also an effective way to create pivot table
s--but here we just use it as a simpler way to see both
data segmentations
print nyt.groupby('Signed_In').describe()
```

```
           Age  Gender    Impressions        Clicks  Signe
d_In
count  137106  137106  137106.000000  137106.00000     13
```

```
7106
mean        0       0      4.999657      0.14208
   0
std         0       0      2.240662      0.38551
   0
min         0       0      0.000000      0.00000
   0
25%         0       0      3.000000      0.00000
   0
50%         0       0      5.000000      0.00000
   0
75%         0       0      6.000000      0.00000
   0
max         0       0     18.000000      4.00000
   0
                           Age          Clicks           Gen
der    Impressions
Signed_In

0         count  137106.000000  137106.000000  137106.000
000  137106.000000
          mean        0.000000       0.142080        0.000
000       4.999657
          std         0.000000       0.385510        0.000
000       2.240662
          min         0.000000       0.000000        0.000
000       0.000000
          25%         0.000000       0.000000        0.000
000       3.000000
          50%         0.000000       0.000000        0.000
000       5.000000
          75%         0.000000       0.000000        0.000
000       6.000000
          max         0.000000       4.000000        0.000
000      18.000000
1         count  321335.000000  321335.000000  321335.000
000  321335.000000
          mean       42.062054       0.071480        0.523
644       5.010584
          std        16.308117       0.268659        0.499
441       2.238784
          min         7.000000       0.000000        0.000
000       0.000000
          25%        29.000000       0.000000        0.000
000       3.000000
```

```
            50%         41.000000       0.000000        1.000
000         5.000000
            75%         53.000000       0.000000        1.000
000         6.000000
            max        108.000000       3.000000        1.000
000        20.000000
```

NEXT STEPS

What's the objective? We want to see if there is a correlation with age, gender, and Signed_Out with click_thru_rate, measured as clicks / Impressions

other ways to fiddle around with pandas (slicing) create a function that defines click through rate for each row

introduction to matplotlib

plot each and everything

In [5]:
```
# Observe what occurs if we divide ints, compared to div
iding floats:
print 1 / 2
print 1.0 / 2.0
```

```
0
0.5
```

In [191]:
```
# in order to create click thru, we need clicks and im
pressions to be floats, otherwise they do not divide a
s a human would expect!

nyt['Clicks'] = nyt['Clicks'].astype('float')
nyt['Impressions'] = nyt['Impressions'].astype('float'
)

# You could also change them at the same time
nyt[['Clicks', 'Impressions']] = nyt[['Clicks', 'Impre
ssions']].astype('float')

# Or pass the columns in as a list varible

columns_to_float=['Clicks', 'Impressions']
nyt[columns_to_float] = nyt[columns_to_float].astype('
float')


nyt['Click_Thru'] = nyt['Clicks'] / nyt['Impressions']
```

In [192]:
```
print nyt.describe()
print nyt.groupby('Signed_In').describe()
```

```
                 Age         Gender    Impressions
  Clicks  \
count  458441.000000  458441.000000  458441.000000  45844
1.000000
mean       29.482551       0.367037       5.007316
0.092594
std        23.607034       0.481997       2.239349
0.309973
min         0.000000       0.000000       0.000000
0.000000
25%         0.000000       0.000000       3.000000
0.000000
50%        31.000000       0.000000       5.000000
0.000000
75%        48.000000       1.000000       6.000000
0.000000
max       108.000000       1.000000      20.000000
4.000000

           Signed_In     Click_Thru
count  458441.000000  455375.000000
```

```
mean         0.700930      0.018471
std          0.457851      0.069034
min          0.000000      0.000000
25%          0.000000      0.000000
50%          1.000000      0.000000
75%          1.000000      0.000000
max          1.000000      1.000000
                           Age      Click_Thru          Cli
cks          Gender  \
Signed_In

0          count  137106.000000  136177.000000  137106.000
000  137106.000000
           mean        0.000000       0.028355       0.142
080        0.000000
           std         0.000000       0.085324       0.385
510        0.000000
           min         0.000000       0.000000       0.000
000        0.000000
           25%         0.000000       0.000000       0.000
000        0.000000
           50%         0.000000       0.000000       0.000
000        0.000000
           75%         0.000000       0.000000       0.000
000        0.000000
           max         0.000000       1.000000       4.000
000        0.000000
1          count  321335.000000  319198.000000  321335.000
000  321335.000000
           mean       42.062054       0.014254       0.071
480        0.523644
           std        16.308117       0.060280       0.268
659        0.499441
           min         7.000000       0.000000       0.000
000        0.000000
           25%        29.000000       0.000000       0.000
000        0.000000
           50%        41.000000       0.000000       0.000
000        1.000000
           75%        53.000000       0.000000       0.000
000        1.000000
           max       108.000000       1.000000       3.000
000        1.000000

                    Impressions
```

```
Signed_In
0          count  137106.000000
           mean        4.999657
           std         2.240662
           min         0.000000
           25%         3.000000
           50%         5.000000
           75%         6.000000
           max        18.000000
1          count  321335.000000
           mean        5.010584
           std         2.238784
           min         0.000000
           25%         3.000000
           50%         5.000000
           75%         6.000000
           max        20.000000
```

In [193]:
```
# Keep in mind you can use lists in a group by as well
print nyt.groupby(['Signed_In', 'Gender']).describe()
```

```
                                    Age     Click_Thru
    Clicks  \
Signed_In Gender

0         0      count  137106.000000  136177.000000  137
106.000000
                 mean        0.000000       0.028355
  0.142080
                 std         0.000000       0.085324
  0.385510
                 min         0.000000       0.000000
  0.000000
                 25%         0.000000       0.000000
  0.000000
                 50%         0.000000       0.000000
  0.000000
                 75%         0.000000       0.000000
  0.000000
                 max         0.000000       1.000000
  4.000000
1         0      count  153070.000000  152052.000000  153
070.000000
                 mean       43.423336       0.014622
```

```
  0.073117
                std      16.763906      0.060956
  0.271194
                min       7.000000      0.000000
  0.000000
                25%      30.000000      0.000000
  0.000000
                50%      42.000000      0.000000
  0.000000
                75%      55.000000      0.000000
  0.000000
                max     108.000000      1.000000
  3.000000
         1      count  168265.000000  167146.000000  168
265.000000
                mean     40.823701      0.013919
  0.069991
                std      15.780505      0.059656
  0.266324
                min       7.000000      0.000000
  0.000000
                25%      28.000000      0.000000
  0.000000
                50%      40.000000      0.000000
  0.000000
                75%      52.000000      0.000000
  0.000000
                max     107.000000      1.000000
  3.000000

                          Impressions
Signed_In Gender
0         0      count  137106.000000
                 mean        4.999657
                 std         2.240662
                 min         0.000000
                 25%         3.000000
                 50%         5.000000
                 75%         6.000000
                 max        18.000000
1         0      count  153070.000000
                 mean        5.012733
                 std         2.238426
                 min         0.000000
                 25%         3.000000
```

```
          50%           5.000000
          75%           6.000000
          max          17.000000
1         count    168265.000000
          mean          5.008629
          std           2.239114
          min           0.000000
          25%           3.000000
          50%           5.000000
          75%           6.000000
          max          20.000000
```

matplotlib

`matplotlib`'s core functionality serves as a plotting tool within python. While calling `.describe()` on DataFrames is useful to get a rough idea of what your data looks like, plots allow you to visualize what your data really looks like.

Consider the following data set and code:

```
In [194]: anscombe = pd.DataFrame({
              'x' :  [10, 8, 13, 9, 11, 14, 6, 4, 12, 7, 5],
              'y1' : [8.04, 6.95, 7.58, 8.81, 8.33, 9.96, 7.24,
          4.26, 10.84, 4.82, 5.68],
              'y2' : [9.14, 8.14, 8.74, 8.77, 9.26, 8.10, 6.13,
          3.10, 9.13, 7.26, 4.74],
              'y3' : [7.46, 6.77, 12.74, 7.11, 7.81, 8.84, 6.08,
           5.39, 8.15, 6.42, 5.73],
              'x4' : [8,8,8,8,8,8,8,19,8,8,8],
              'y4' : [6.58,5.76,7.71,8.84,8.47,7.04,5.25,12.50,5
          .56,7.91,6.89],
          })

          anscombe.describe()
```

Out[194]:

	x	x4	y1	y2	y3	y4
count	11.000000	11.000000	11.000000	11.000000	11.000000	11.0000
mean	9.000000	9.000000	7.500909	7.500909	7.500000	7.50090
std	3.316625	3.316625	2.031568	2.031657	2.030424	2.03057
min	4.000000	8.000000	4.260000	3.100000	5.390000	5.25000
25%	6.500000	8.000000	6.315000	6.695000	6.250000	6.17000
50%	9.000000	8.000000	7.580000	8.140000	7.110000	7.04000
75%	11.500000	8.000000	8.570000	8.950000	7.980000	8.19000
max	14.000000	19.000000	10.840000	9.260000	12.740000	12.5000

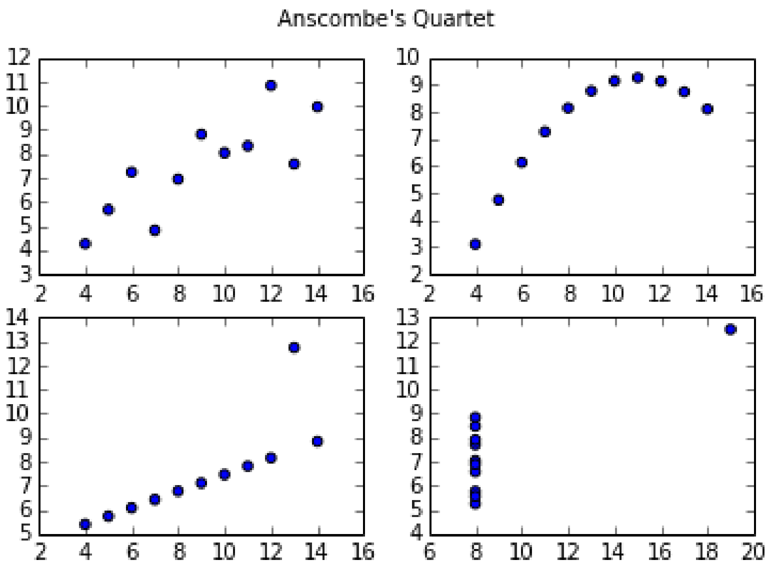
Visually from creating the data frame you can tell the data looks different, yet in the `.describe()` call, the data shares very similar features. The two primary plotting tools we uses from matplotlib are histograms and scatterplots, which help us understand the shape of data.

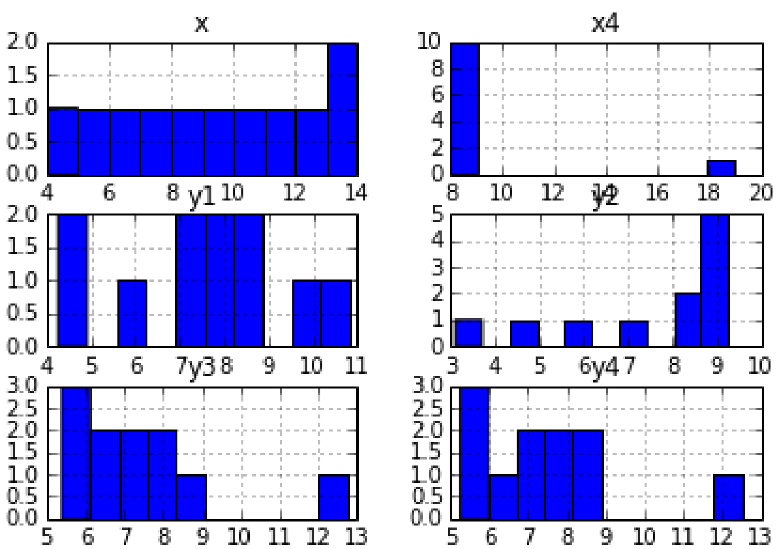
```
In [195]: from matplotlib import pylab as plt

          fig = plt.figure()
          ax1 = fig.add_subplot(2,2,1) # Two rows, Two columns, 
          First Plot
          ax1.scatter(anscombe.x, anscombe.y1)
          ax2 = fig.add_subplot(2,2,2) # Second Plot (Plots go L
           -> R, Up -> Down)
          ax2.scatter(anscombe.x, anscombe.y2)
          ax3 = fig.add_subplot(2,2,3) # Third
          ax3.scatter(anscombe.x, anscombe.y3)
          ax4 = fig.add_subplot(2,2,4) # Fourth
          ax4.scatter(anscombe.x4, anscombe.y4)
          fig.suptitle("Anscombe's Quartet")
          print fig

          # Pandas also has matplotlib built in, which is incred
          ibly useful for creating fast histograms.
          print anscombe.hist()
```

```
Figure(480x320)
[[<matplotlib.axes.AxesSubplot object at 0x10b9b1950>
  <matplotlib.axes.AxesSubplot object at 0x1108df110>]
 [<matplotlib.axes.AxesSubplot object at 0x10dafce90>
  <matplotlib.axes.AxesSubplot object at 0x10bd07950>]
 [<matplotlib.axes.AxesSubplot object at 0x10bb874d0>
  <matplotlib.axes.AxesSubplot object at 0x10baa8550>]]
```



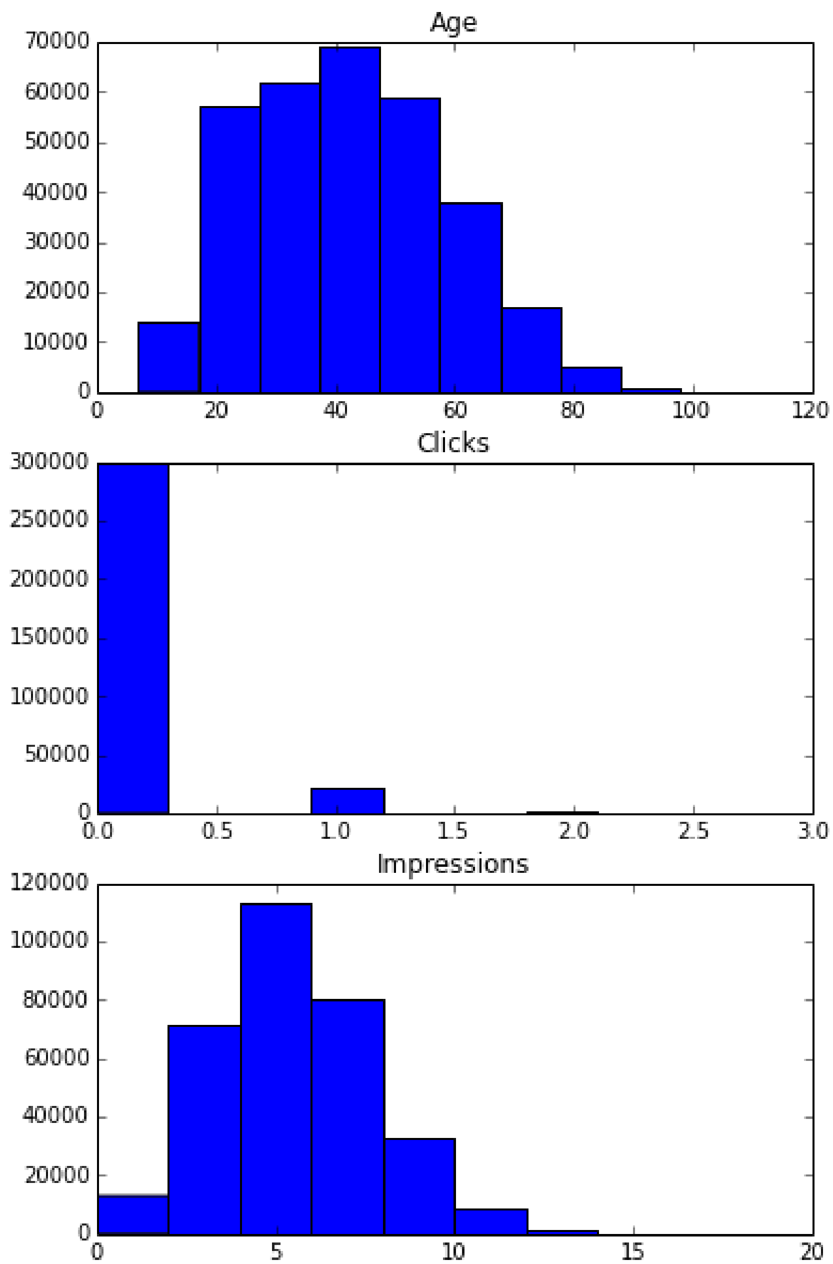


Use these new tools in order to visualize some of this New York Times ad performance data.

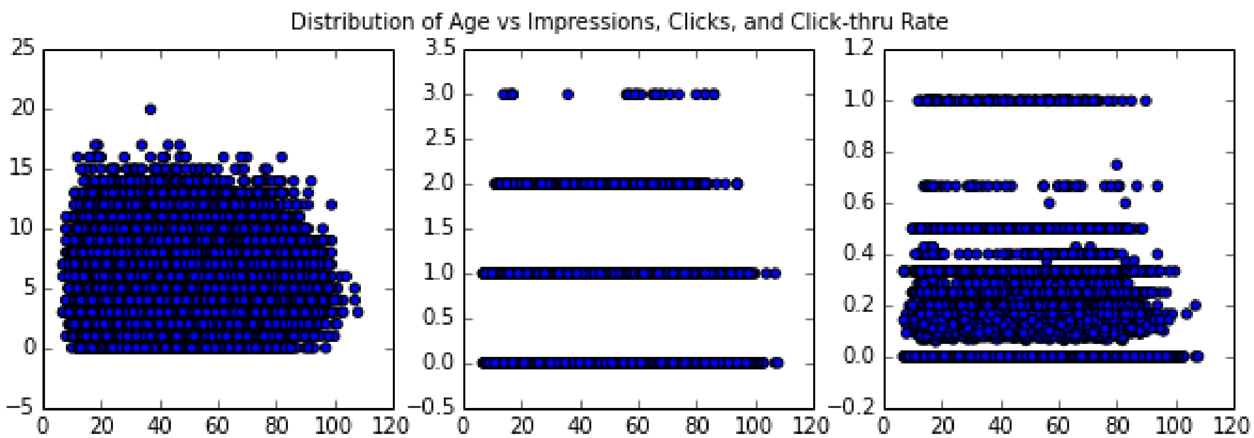
In [196]:
```
nyt_signed_in = nyt[nyt['Signed_In'] ==1]

fig = plt.figure(figsize=(6, 10))
ax1 = fig.add_subplot(3, 1, 1)

ax1.set_title('Age')
ax1.hist(nyt_signed_in['Age'])
ax2 = fig.add_subplot(3, 1, 2)
ax2.set_title('Clicks')
ax2.hist(nyt_signed_in['Clicks'])
ax3 = fig.add_subplot(3, 1, 3)
ax3.set_title('Impressions')
ax3.hist(nyt_signed_in['Impressions'])
fig.show()
```



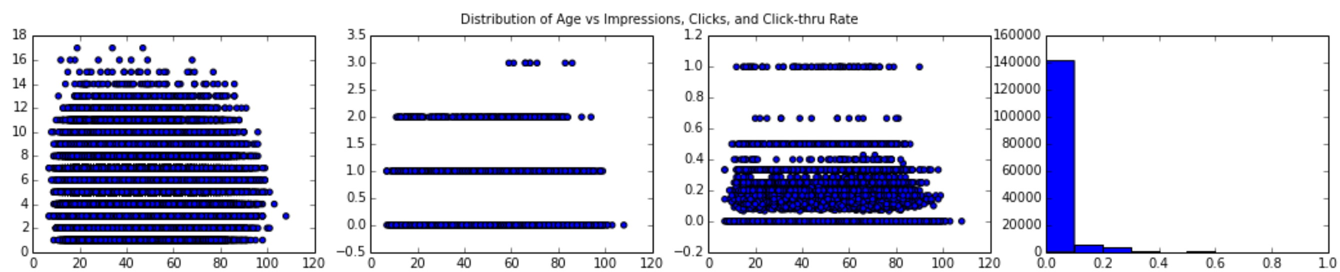
In [197]:
```
fig = plt.figure(figsize=(10, 3))
fig.suptitle('Distribution of Age vs Impressions, Clic
ks, and Click-thru Rate')
ax1 = fig.add_subplot(131)
ax1.scatter(nyt_signed_in['Age'], nyt_signed_in['Impre
ssions'])
ax2 = fig.add_subplot(132)
ax2.scatter(nyt_signed_in['Age'], nyt_signed_in['Click
s'])
ax3 = fig.add_subplot(133)
ax3.scatter(nyt_signed_in['Age'], nyt_signed_in['Click
_Thru'])
fig.show()
```



Even though we see some clear normal-like distributions Why do we care about normal distributions? (http://www.quora.com/Normal-Distribution-statistics/Why-do-we-use-the-normal-distribution) with histograms, Comparing Age to a few variables does not show any clear relationships.

In [198]:
```
import numpy as np
# Just in case, let's compare just one gender:
nyt_gender0 = nyt_signed_in[nyt_signed_in['Gender'] ==
 0]
nyt_gender0 = nyt_gender0[np.isfinite(nyt_gender0['Cli
ck_Thru'])]
fig = plt.figure(figsize=(18, 3))
fig.suptitle('Distribution of Age vs Impressions, Clic
ks, and Click-thru Rate')
ax1 = fig.add_subplot(141)
ax1.scatter(nyt_gender0['Age'], nyt_gender0['Impressio
ns'])

ax2 = fig.add_subplot(142)
ax2.scatter(nyt_gender0['Age'], nyt_gender0['Clicks'])
ax3 = fig.add_subplot(143)
ax3.scatter(nyt_gender0['Age'], nyt_gender0['Click_Thr
u'])
ax4 = fig.add_subplot(144)
ax4.hist(nyt_gender0['Click_Thru'])
fig.show()
```



scikit-learn

`Scikit-learn` (often 'sklearn') is one of several core machine learning packages available in python.

`scikit-learn` is designed to be modular. Many parts of the libraries are super classes of base packages, which means that many of them share the same functionality. For example, consider making the following classes, Array and Matrix, where Matrix is actually a special kind of Array:

In [199]:
```
class MyArray():

    def __init__(self, x, y, dim, value):
        self.x = x
        self.y = y
        self.dim = dim
        self.value = value

    def multi(self):
        values = []
        for i in range(self.x):
            values.append([[self.value for k in xrange
(self.y)] for j in xrange(self.dim)])
        return values

    def show(self):
        print "Showing", self.__class__.__name__
        for i in self.multi():
            print i

my_array = MyArray(5, 8, 5, 0)
my_array.show()

print
class MyMatrix(MyArray):
    def __init__(self, x, y, value):
        self.x = x
        self.y = y
        self.dim = 1
        self.value = value

my_matrix = MyMatrix(4, 8, 0)
my_matrix.show()
```

```
Showing MyArray
[[0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0,
0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0,
 0, 0, 0, 0, 0]]
[[0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0,
0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0,
 0, 0, 0, 0, 0]]
[[0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0,
0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0,
 0, 0, 0, 0, 0]]
[[0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0,
0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0,
 0, 0, 0, 0, 0]]
[[0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0,
0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0,
 0, 0, 0, 0, 0]]

Showing MyMatrix
[[0, 0, 0, 0, 0, 0, 0, 0]]
[[0, 0, 0, 0, 0, 0, 0, 0]]
[[0, 0, 0, 0, 0, 0, 0, 0]]
[[0, 0, 0, 0, 0, 0, 0, 0]]
```

Notice how the MyMatrix class borrows functions from the MyArray class? The only difference is that Matrices are special kinds of Arrays, in that they are only 2 dimensional.

(note, a better way to do this would be to use the `numpy.array()`)

`scikit-learn` is in fact, very similar, as most functionality is built on a primary `.fit()` function for each learning algorithm (even their dummy regression, which is described here (http://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyRegressor.html), follows the same base Class!)

In [200]:
```
from sklearn.dummy import DummyRegressor
from sklearn.linear_model import LinearRegression

nyt = nyt[np.isfinite(nyt['Click_Thru'])]
dummy_fit = DummyRegressor().fit(nyt[['Signed_In', 'Ge
nder', 'Age']], nyt['Click_Thru'])
lm_fit = LinearRegression().fit(nyt[['Signed_In', 'Gen
der', 'Age']], nyt['Click_Thru'])
anscombe_fit1 = LinearRegression().fit(anscombe[['x']]
, anscombe['y1'])
anscombe_fit2 = LinearRegression().fit(anscombe[['x']]
, anscombe['y2'])
anscombe_fit3 = LinearRegression().fit(anscombe[['x']]
, anscombe['y3'])
anscombe_fit4 = LinearRegression().fit(anscombe[['x4']
], anscombe['y4'])
```

In [201]:
```
print "Another point of weakness: Anscombe's Quartet h
as the same linear model for each group"
print anscombe_fit1.coef_, anscombe_fit1.intercept_
print anscombe_fit2.coef_, anscombe_fit2.intercept_
print anscombe_fit3.coef_, anscombe_fit3.intercept_
print anscombe_fit4.coef_, anscombe_fit4.intercept_

print
print dummy_fit.score(nyt[['Signed_In', 'Gender', 'Age
']], nyt['Click_Thru'])
print lm_fit.score(nyt[['Signed_In', 'Gender', 'Age']]
, nyt['Click_Thru'])
```

```
Another point of weakness: Anscombe's Quartet has the sam
e linear model for each group
[ 0.50009091] 3.00009090909
[ 0.5] 3.00090909091
[ 0.49972727] 3.00245454545
[ 0.49990909] 3.00172727273

0.0
0.0103350341913
```

Above, the `.score()` function, another common function across many `scikit-learn` Classes, returns back the r-squared value for this linear model. Unfortunately, it does not seem like you can fit the nytimes data linearly, as the score is very close to 0.

Datasets available within `scikit-learn`

There are also several data sets available within `scikit-learn`, however they primarily rely on the numpy objects instead. Become familiar with Fisher's Iris Data set, as it's a fairly common and unique data set, and will likely be referenced in the GA's primary data science class, or other classes you make take in the future.

In [153]:
```
from sklearn import datasets

iris = datasets.load_iris()

print iris.DESCR
```

```
Iris Plants Database

Notes
-----
Data Set Characteristics:
    :Number of Instances: 150 (50 in each of three classe
s)
    :Number of Attributes: 4 numeric, predictive attribut
es and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica
    :Summary Statistics:
    ============== ==== ==== ======= ===== ==============
```

```
======
                    Min  Max   Mean    SD   Class Correla
tion
    ============== ==== ==== ======= ===== ==============
======
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (hi
gh!)
    petal width:    0.1  2.5   1.20  0.76     0.9565  (hi
gh!)
    ============== ==== ==== ======= ===== ==============
======
    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.go
v)
    :Date: July, 1988

This is a copy of UCI ML iris datasets.
http://archive.ics.uci.edu/ml/datasets/Iris

The famous Iris database, first used by Sir R.A Fisher

This is perhaps the best known database to be found in th
e
pattern recognition literature.  Fisher's paper is a clas
sic in the field and
is referenced frequently to this day.  (See Duda & Hart,
for example.)  The
data set contains 3 classes of 50 instances each, where e
ach class refers to a
type of iris plant.  One class is linearly separable from
 the other 2; the
latter are NOT linearly separable from each other.

References
----------
   - Fisher,R.A. "The use of multiple measurements in tax
onomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in
 "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda,R.O., & Hart,P.E. (1973) Pattern Classification
```

```
 and Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.
See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborho
od: A New System
     Structure and Classification Rule for Recognition in
 Partially Exposed
     Environments".  IEEE Transactions on Pattern Analysi
s and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rul
e".  IEEE Transactions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et
 al"s AUTOCLASS II
     conceptual clustering system finds 3 classes in the
data.
   - Many, many more ...
```

In [154]:
```
print iris.data
```

```
[[ 5.1  3.5  1.4  0.2]
 [ 4.9  3.   1.4  0.2]
 [ 4.7  3.2  1.3  0.2]
 [ 4.6  3.1  1.5  0.2]
 [ 5.   3.6  1.4  0.2]
 [ 5.4  3.9  1.7  0.4]
 [ 4.6  3.4  1.4  0.3]
 [ 5.   3.4  1.5  0.2]
 [ 4.4  2.9  1.4  0.2]
 [ 4.9  3.1  1.5  0.1]
 [ 5.4  3.7  1.5  0.2]
 [ 4.8  3.4  1.6  0.2]
 [ 4.8  3.   1.4  0.1]
 [ 4.3  3.   1.1  0.1]
 [ 5.8  4.   1.2  0.2]
 [ 5.7  4.4  1.5  0.4]
 [ 5.4  3.9  1.3  0.4]
 [ 5.1  3.5  1.4  0.3]
 [ 5.7  3.8  1.7  0.3]
 [ 5.1  3.8  1.5  0.3]
 [ 5.4  3.4  1.7  0.2]
 [ 5.1  3.7  1.5  0.4]
```

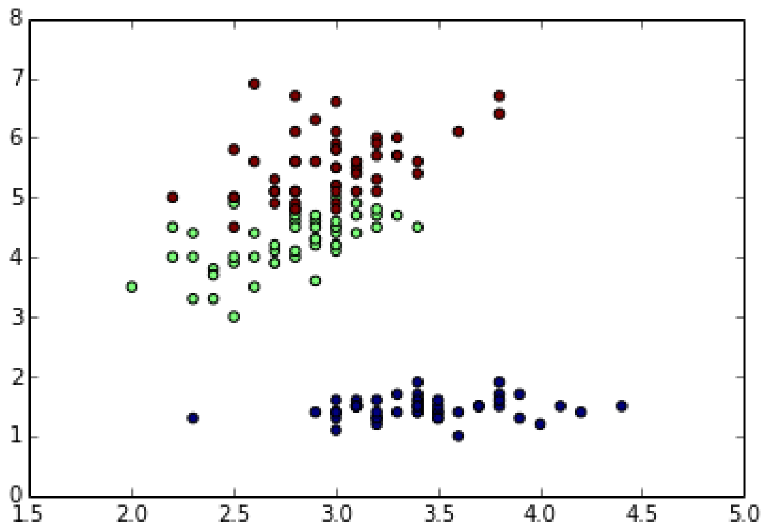
```
[ 4.6  3.6  1.   0.2]
[ 5.1  3.3  1.7  0.5]
[ 4.8  3.4  1.9  0.2]
[ 5.   3.   1.6  0.2]
[ 5.   3.4  1.6  0.4]
[ 5.2  3.5  1.5  0.2]
[ 5.2  3.4  1.4  0.2]
[ 4.7  3.2  1.6  0.2]
[ 4.8  3.1  1.6  0.2]
[ 5.4  3.4  1.5  0.4]
[ 5.2  4.1  1.5  0.1]
[ 5.5  4.2  1.4  0.2]
[ 4.9  3.1  1.5  0.1]
[ 5.   3.2  1.2  0.2]
[ 5.5  3.5  1.3  0.2]
[ 4.9  3.1  1.5  0.1]
[ 4.4  3.   1.3  0.2]
[ 5.1  3.4  1.5  0.2]
[ 5.   3.5  1.3  0.3]
[ 4.5  2.3  1.3  0.3]
[ 4.4  3.2  1.3  0.2]
[ 5.   3.5  1.6  0.6]
[ 5.1  3.8  1.9  0.4]
[ 4.8  3.   1.4  0.3]
[ 5.1  3.8  1.6  0.2]
[ 4.6  3.2  1.4  0.2]
[ 5.3  3.7  1.5  0.2]
[ 5.   3.3  1.4  0.2]
[ 7.   3.2  4.7  1.4]
[ 6.4  3.2  4.5  1.5]
[ 6.9  3.1  4.9  1.5]
[ 5.5  2.3  4.   1.3]
[ 6.5  2.8  4.6  1.5]
[ 5.7  2.8  4.5  1.3]
[ 6.3  3.3  4.7  1.6]
[ 4.9  2.4  3.3  1. ]
[ 6.6  2.9  4.6  1.3]
[ 5.2  2.7  3.9  1.4]
[ 5.   2.   3.5  1. ]
[ 5.9  3.   4.2  1.5]
[ 6.   2.2  4.   1. ]
[ 6.1  2.9  4.7  1.4]
[ 5.6  2.9  3.6  1.3]
[ 6.7  3.1  4.4  1.4]
[ 5.6  3.   4.5  1.5]
```

```
[ 5.8  2.7  4.1  1. ]
[ 6.2  2.2  4.5  1.5]
[ 5.6  2.5  3.9  1.1]
[ 5.9  3.2  4.8  1.8]
[ 6.1  2.8  4.   1.3]
[ 6.3  2.5  4.9  1.5]
[ 6.1  2.8  4.7  1.2]
[ 6.4  2.9  4.3  1.3]
[ 6.6  3.   4.4  1.4]
[ 6.8  2.8  4.8  1.4]
[ 6.7  3.   5.   1.7]
[ 6.   2.9  4.5  1.5]
[ 5.7  2.6  3.5  1. ]
[ 5.5  2.4  3.8  1.1]
[ 5.5  2.4  3.7  1. ]
[ 5.8  2.7  3.9  1.2]
[ 6.   2.7  5.1  1.6]
[ 5.4  3.   4.5  1.5]
[ 6.   3.4  4.5  1.6]
[ 6.7  3.1  4.7  1.5]
[ 6.3  2.3  4.4  1.3]
[ 5.6  3.   4.1  1.3]
[ 5.5  2.5  4.   1.3]
[ 5.5  2.6  4.4  1.2]
[ 6.1  3.   4.6  1.4]
[ 5.8  2.6  4.   1.2]
[ 5.   2.3  3.3  1. ]
[ 5.6  2.7  4.2  1.3]
[ 5.7  3.   4.2  1.2]
[ 5.7  2.9  4.2  1.3]
[ 6.2  2.9  4.3  1.3]
[ 5.1  2.5  3.   1.1]
[ 5.7  2.8  4.1  1.3]
[ 6.3  3.3  6.   2.5]
[ 5.8  2.7  5.1  1.9]
[ 7.1  3.   5.9  2.1]
[ 6.3  2.9  5.6  1.8]
[ 6.5  3.   5.8  2.2]
[ 7.6  3.   6.6  2.1]
[ 4.9  2.5  4.5  1.7]
[ 7.3  2.9  6.3  1.8]
[ 6.7  2.5  5.8  1.8]
[ 7.2  3.6  6.1  2.5]
[ 6.5  3.2  5.1  2. ]
[ 6.4  2.7  5.3  1.9]
```

```
 [ 6.8  3.   5.5  2.1]
 [ 5.7  2.5  5.   2. ]
 [ 5.8  2.8  5.1  2.4]
 [ 6.4  3.2  5.3  2.3]
 [ 6.5  3.   5.5  1.8]
 [ 7.7  3.8  6.7  2.2]
 [ 7.7  2.6  6.9  2.3]
 [ 6.   2.2  5.   1.5]
 [ 6.9  3.2  5.7  2.3]
 [ 5.6  2.8  4.9  2. ]
 [ 7.7  2.8  6.7  2. ]
 [ 6.3  2.7  4.9  1.8]
 [ 6.7  3.3  5.7  2.1]
 [ 7.2  3.2  6.   1.8]
 [ 6.2  2.8  4.8  1.8]
 [ 6.1  3.   4.9  1.8]
 [ 6.4  2.8  5.6  2.1]
 [ 7.2  3.   5.8  1.6]
 [ 7.4  2.8  6.1  1.9]
 [ 7.9  3.8  6.4  2. ]
 [ 6.4  2.8  5.6  2.2]
 [ 6.3  2.8  5.1  1.5]
 [ 6.1  2.6  5.6  1.4]
 [ 7.7  3.   6.1  2.3]
 [ 6.3  3.4  5.6  2.4]
 [ 6.4  3.1  5.5  1.8]
 [ 6.   3.   4.8  1.8]
 [ 6.9  3.1  5.4  2.1]
 [ 6.7  3.1  5.6  2.4]
 [ 6.9  3.1  5.1  2.3]
 [ 5.8  2.7  5.1  1.9]
 [ 6.8  3.2  5.9  2.3]
 [ 6.7  3.3  5.7  2.5]
 [ 6.7  3.   5.2  2.3]
 [ 6.3  2.5  5.   1.9]
 [ 6.5  3.   5.2  2. ]
 [ 6.2  3.4  5.4  2.3]
 [ 5.9  3.   5.1  1.8]]
```

In [162]:
```
print plt.scatter(iris.data[:,1], iris.data[:,2], c=iris.target)
```

```
<matplotlib.collections.PathCollection object at 0x10b8a43d0>
```



Next Steps

Practice doing much of the same functionality we did today with three different data sets from here (http://vincentarelbundock.github.io/Rdatasets/datasets.html).

Primarily, your goals are to: * Show how to load a csv file into python * Use pandas to understand the numerical portions of the data * Use matplotlib to visualize the data * Use scikit-learn to fit the data to some dependent feature (y)